Soundness and Completeness of an Axiomatic System for First-Order Logic

Asta Halkjær From

March 17, 2025

Abstract

This work is a formalization of the soundness and completeness of an axiomatic system for first-order logic. The proof system is based on System Q1 by Smullyan and the completeness proof follows his textbook "First-Order Logic" (Springer-Verlag 1968) [2]. The completeness proof is in the Henkin style [1] where a consistent set is extended to a maximal consistent set using Lindenbaum's construction and Henkin witnesses are added during the construction to ensure saturation as well. The resulting set is a Hintikka set which, by the model existence theorem, is satisfiable in the Herbrand universe.

Contents

theory *FOL-Axiomatic* **imports** *HOL−Library.Countable* **begin**

1 Syntax

datatype (*params-tm*: $'f$) *tm*
  = *Var nat* (‹#›)
  | *Fun* $'f$ ‹$'f$ *tm list*› (‹†›)

abbreviation *Const* (‹⋆›) **where** ‹⋆*a* ≡ †*a* []›

datatype (*params-fm*: $'f$, $'p$) *fm*
  = *Falsity* (‹⊥›)
  | *Pre* $'p$ ‹$'f$ *tm list*› (‹‡›)
  | *Imp* ‹($'f$, $'p$) *fm*› ‹($'f$, $'p$) *fm*› (**infixr** ‹⟶› *55*)
  | *Uni* ‹($'f$, $'p$) *fm*› (‹∀›)

abbreviation *Neg* (‹¬ -› [*70*] *70*) **where** ‹¬ *p* ≡ *p* ⟶ ⊥›

term ‹∀ (⊥ ⟶ ‡''*P*'' [†''*f*'' [#*0*]])›

2 Semantics

definition *shift* (‹-⟨-:-⟩›) **where**
  ‹*E*⟨*n*:*x*⟩ *m* ≡ *if m* < *n then E m else if m* = *n then x else E* (*m*−*1*)›

primrec *semantics-tm* (‹⦇-, -⦈›) **where**
  ‹⦇*E*, *F*⦈ (#*n*) = *E n*›
| ‹⦇*E*, *F*⦈ (†*f ts*) = *F f* (*map* ⦇*E*, *F*⦈ *ts*)›

primrec *semantics-fm* (‹⟦-, -, -⟧›) **where**
  ‹⟦-, -, -⟧ ⊥ = *False*›
| ‹⟦*E*, *F*, *G*⟧ (‡*P ts*) = *G P* (*map* ⦇*E*, *F*⦈ *ts*)›
| ‹⟦*E*, *F*, *G*⟧ (*p* ⟶ *q*) = (⟦*E*, *F*, *G*⟧ *p* ⟶ ⟦*E*, *F*, *G*⟧ *q*)›
| ‹⟦*E*, *F*, *G*⟧ (∀ *p*) = (∀ *x*. ⟦*E*⟨*0*:*x*⟩, *F*, *G*⟧ *p*)›

proposition ‹⟦*E*, *F*, *G*⟧ (∀ (‡*P* [# *0*]) ⟶ ‡*P* [⋆*a*])›
  by (*simp add*: *shift-def*)

3 Operations

3.1 Shift

context fixes *n m* :: *nat* **begin**

lemma *shift-eq* [*simp*]: ‹*n* = *m* ⟹ *E*⟨*n*:*x*⟩ *m* = *x*›
  by (*simp add*: *shift-def*)

lemma *shift-gt* [*simp*]: ‹$m < n \Longrightarrow E\langle n{:}x\rangle\ m = E\ m$›
  by (*simp add*: *shift-def*)

lemma *shift-lt* [*simp*]: ‹$n < m \Longrightarrow E\langle n{:}x\rangle\ m = E\ (m-1)$›
  by (*simp add*: *shift-def*)

lemma *shift-commute* [*simp*]: ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle) = (E\langle 0{:}x\rangle\langle n+1{:}y\rangle)$›
proof
  fix m
  show ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle)\ m = (E\langle 0{:}x\rangle\langle n+1{:}y\rangle)\ m$›
    unfolding *shift-def* **by** (*cases m*) *simp-all*
qed

end

3.2 Parameters

abbreviation ‹$params\ S \equiv \bigcup p \in S.\ params\text{-}fm\ p$›

lemma *upd-params-tm* [*simp*]: ‹$f \notin params\text{-}tm\ t \Longrightarrow (\!|E,\ F(f := x)|\!)\ t = (\!|E,\ F|\!)\ t$›
  by (*induct t*) (*auto cong*: *map-cong*)

lemma *upd-params-fm* [*simp*]: ‹$f \notin params\text{-}fm\ p \Longrightarrow [\![E,\ F(f := x),\ G]\!]\ p = [\![E,\ F,\ G]\!]\ p$›
  by (*induct p arbitrary*: *E*) (*auto cong*: *map-cong*)

lemma *finite-params-tm* [*simp*]: ‹*finite* (*params-tm t*)›
  by (*induct t*) *simp-all*

lemma *finite-params-fm* [*simp*]: ‹*finite* (*params-fm p*)›
  by (*induct p*) *simp-all*

3.3 Instantiation

primrec *lift-tm* (‹$\Uparrow$›) **where**
  ‹$\Uparrow(\#n) = \#(n+1)$›
| ‹$\Uparrow(\dagger f\ ts) = \dagger f\ (map\ \Uparrow\ ts)$›

primrec *inst-tm* (‹$\langle\!\langle$-'/-$\rangle\!\rangle$›) **where**
  ‹$\langle\!\langle s/m\rangle\!\rangle(\#n) = (if\ n < m\ then\ \#n\ else\ if\ n = m\ then\ s\ else\ \#(n-1))$›
| ‹$\langle\!\langle s/m\rangle\!\rangle(\dagger f\ ts) = \dagger f\ (map\ \langle\!\langle s/m\rangle\!\rangle\ ts)$›

primrec *inst-fm* (‹$\langle$-'/-$\rangle$›) **where**
  ‹$\langle$-/-$\rangle\bot = \bot$›
| ‹$\langle s/m\rangle(\ddagger P\ ts) = \ddagger P\ (map\ \langle\!\langle s/m\rangle\!\rangle\ ts)$›
| ‹$\langle s/m\rangle(p \longrightarrow q) = \langle s/m\rangle p \longrightarrow \langle s/m\rangle q$›
| ‹$\langle s/m\rangle(\forall\ p) = \forall\ (\langle\Uparrow s/m+1\rangle p)$›

lemma *lift-lemma* [*simp*]: ‹$(\!|E\langle 0{:}x\rangle,\ F|\!)$ $(\uparrow t)$ = $(\!|E,\ F|\!)$ *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *inst-tm-semantics* [*simp*]: ‹$(\!|E,\ F|\!)$ $(\langle\!\langle s/m\rangle\!\rangle t)$ = $(\!|E\langle m{:}(\!|E,\ F|\!)\ s\rangle,\ F|\!)$ *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *inst-fm-semantics* [*simp*]: ‹$[\![E,\ F,\ G]\!]$ $(\langle t/m\rangle p)$ = $[\![E\langle m{:}(\!|E,\ F|\!)\ t\rangle,\ F,\ G]\!]$ *p*›
by (*induct p arbitrary*: *E m t*) (*auto cong*: *map-cong*)

3.4 Size

The built-in *size* is not invariant under substitution.

primrec *size-fm* **where**
‹*size-fm* $\bot$ = *1*›
| ‹*size-fm* ($\ddagger$- -) = *1*›
| ‹*size-fm* ($p \longrightarrow q$) = *1* + *size-fm p* + *size-fm q*›
| ‹*size-fm* ($\forall\, p$) = *1* + *size-fm p*›

lemma *size-inst-fm* [*simp*]: ‹*size-fm* $(\langle t/m\rangle p)$ = *size-fm p*›
by (*induct p arbitrary*: *m t*) *simp-all*

4 Propositional Semantics

primrec *boolean* **where**
‹*boolean* - - $\bot$ = *False*›
| ‹*boolean G* - ($\ddagger P$ *ts*) = *G P ts*›
| ‹*boolean G A* ($p \longrightarrow q$) = (*boolean G A p* $\longrightarrow$ *boolean G A q*)›
| ‹*boolean* - *A* ($\forall\, p$) = *A* ($\forall\, p$)›

abbreviation ‹*tautology p* $\equiv \forall$ *G A*. *boolean G A p*›

proposition ‹*tautology* ($\forall$ ($\ddagger P$ [$\#0$]) $\longrightarrow$ $\forall$ ($\ddagger P$ [$\#0$]))›
by *simp*

lemma *boolean-semantics*: ‹*boolean* (λa. *G a* $\circ$ *map* $(\!|E,\ F|\!)$) $[\![E,\ F,\ G]\!]$ = $[\![E,\ F,\ G]\!]$›
proof
fix *p*
show ‹*boolean* (λa. *G a* $\circ$ *map* $(\!|E,\ F|\!)$) $[\![E,\ F,\ G]\!]$ *p* = $[\![E,\ F,\ G]\!]$ *p*›
by (*induct p*) *simp-all*
qed

lemma *tautology*[*simp*]: ‹*tautology p* $\Longrightarrow$ $[\![E,\ F,\ G]\!]$ *p*›
using *boolean-semantics* **by** *metis*

proposition ‹$\exists\, p$. ($\forall$ *E F G*. $[\![E,\ F,\ G]\!]$ *p*) $\wedge$ $\neg$ *tautology p*›
by (*metis boolean.simps*(*4*) *fm.simps*(*36*) *semantics-fm.simps*(*1*,*3*,*4*))

5 Calculus

Adapted from System Q1 by Smullyan in First-Order Logic (1968).

inductive *Axiomatic* (‹⊢ -› [*50*] *50*) **where**
  TA: ‹*tautology p* ⟹ ⊢ *p*›
| *IA*: ‹⊢ ∀ *p* ⟶ ⟨*t*/*0*⟩*p*›
| *MP*: ‹⊢ *p* ⟶ *q* ⟹ ⊢ *p* ⟹ ⊢ *q*›
| *GR*: ‹⊢ *q* ⟶ ⟨⋆*a*/*0*⟩*p* ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ ∀ *p*›

We simulate assumptions on the lhs of ⊢ with a chain of implications on the rhs.

primrec *imply* (**infixr** ‹⇝› *56*) **where**
  ‹([] ⇝ *q*) = *q*›
| ‹(*p* # *ps* ⇝ *q*) = (*p* ⟶ *ps* ⇝ *q*)›

abbreviation *Axiomatic-assms* (‹- ⊢ -› [*50*, *50*] *50*) **where**
  ‹*ps* ⊢ *q* ≡ ⊢ *ps* ⇝ *q*›

6 Soundness

theorem *soundness*: ‹⊢ *p* ⟹ ⟦*E*, *F*, *G*⟧ *p*›
proof (*induct p arbitrary*: *F rule*: *Axiomatic.induct*)
  case (*GR q a p*)
  moreover from *this* **have** ‹⟦*E*, *F*(*a* := *x*), *G*⟧ (*q* ⟶ ⟨⋆*a*/*0*⟩*p*)› **for** *x*
    by *blast*
  ultimately show *?case*
    by *fastforce*
qed *auto*

corollary ‹¬ (⊢ ⊥)›
  using *soundness* **by** *fastforce*

7 Derived Rules

lemma *Imp1*: ‹⊢ *q* ⟶ *p* ⟶ *q*›
  and *Imp2*: ‹⊢ (*p* ⟶ *q* ⟶ *r*) ⟶ (*p* ⟶ *q*) ⟶ *p* ⟶ *r*›
  and *Neg*: ‹⊢ ¬ ¬ *p* ⟶ *p*›
  by (*auto intro*: *TA*)

The tautology axiom TA is not used directly beyond this point.

lemma *Tran'*: ‹⊢ (*q* ⟶ *r*) ⟶ (*p* ⟶ *q*) ⟶ *p* ⟶ *r*›
  by (*meson Imp1 Imp2 MP*)

lemma *Swap*: ‹⊢ (*p* ⟶ *q* ⟶ *r*) ⟶ *q* ⟶ *p* ⟶ *r*›
  by (*meson Imp1 Imp2 Tran' MP*)

lemma *Tran*: ‹⊢ (*p* ⟶ *q*) ⟶ (*q* ⟶ *r*) ⟶ *p* ⟶ *r*›

by (*meson Swap Tran' MP*)

Note that contraposition in the other direction is an instance of the lemma Tran.

lemma *contraposition*: ‹$\vdash (\neg\ q \longrightarrow \neg\ p) \longrightarrow p \longrightarrow q$›
by (*meson Neg Swap Tran MP*)

lemma *GR'*: ‹$\vdash \neg\ \langle\star a/0\rangle p \longrightarrow q \Longrightarrow a \notin params\ \{p,\ q\} \Longrightarrow \vdash \neg\ (\forall\ p) \longrightarrow q$›
proof –
assume $*$: ‹$\vdash \neg\ \langle\star a/0\rangle p \longrightarrow q$› **and** *a*: ‹$a \notin params\ \{p,\ q\}$›
have ‹$\vdash \neg\ q \longrightarrow \neg\ \neg\ \langle\star a/0\rangle p$›
using $*$ *Tran MP* **by** *metis*
then have ‹$\vdash \neg\ q \longrightarrow \langle\star a/0\rangle p$›
using *Neg Tran MP* **by** *metis*
then have ‹$\vdash \neg\ q \longrightarrow \forall\ p$›
using *a* **by** (*auto intro*: *GR*)
then have ‹$\vdash \neg\ (\forall\ p) \longrightarrow \neg\ \neg\ q$›
using *Tran MP* **by** *metis*
then show *?thesis*
using *Neg Tran MP* **by** *metis*
qed

lemma *imply-ImpE*: ‹$\vdash ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$›
proof (*induct ps*)
case *Nil*
then show *?case*
by (*metis Imp1 Swap MP imply.simps*(*1*))
next
case (*Cons r ps*)
have ‹$\vdash (r \longrightarrow ps \rightsquigarrow p) \longrightarrow r \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$›
by (*meson Cons.hyps Imp1 Imp2 MP*)
then have ‹$\vdash (r \longrightarrow ps \rightsquigarrow p) \longrightarrow (r \longrightarrow ps \rightsquigarrow (p \longrightarrow q)) \longrightarrow r \longrightarrow ps \rightsquigarrow q$›
by (*meson Imp1 Imp2 MP*)
then show *?case*
by *simp*
qed

lemma *MP'*: ‹$ps \vdash p \longrightarrow q \Longrightarrow ps \vdash p \Longrightarrow ps \vdash q$›
using *imply-ImpE MP* **by** *metis*

lemma *imply-Cons* [*intro*]: ‹$ps \vdash q \Longrightarrow p\ \#\ ps \vdash q$›
by (*auto intro*: *MP Imp1*)

lemma *imply-head* [*intro*]: ‹$p\ \#\ ps \vdash p$›
by (*induct ps*) (*metis Imp1 Imp2 MP imply.simps*, *metis Imp1 Imp2 MP imply.simps*(*2*))

lemma *add-imply* [*simp*]: ‹$\vdash q \Longrightarrow ps \vdash q$›
using *imply-head* **by** (*metis MP imply.simps*(*2*))

lemma *imply-mem* [*simp*]: ‹$p \in$ *set ps* $\Longrightarrow$ *ps* $\vdash p$›
  using *imply-head imply-Cons* **by** (*induct ps*) *fastforce+*

lemma *deduct1*: ‹*ps* $\vdash p \longrightarrow q \Longrightarrow p$ # *ps* $\vdash q$›
  by (*meson MP′ imply-Cons imply-head*)

lemma *imply-append* [*iff*]: ‹(*ps* @ *qs* $\rightsquigarrow r$) = (*ps* $\rightsquigarrow$ *qs* $\rightsquigarrow r$)›
  by (*induct ps*) *simp-all*

lemma *imply-swap-append*: ‹*ps* @ *qs* $\vdash r \Longrightarrow$ *qs* @ *ps* $\vdash r$›
proof (*induct qs arbitrary*: *ps*)
  case *Cons*
  then show *?case*
    by (*metis deduct1 imply.simps*(*2*) *imply-append*)
qed *simp*

lemma *deduct2*: ‹p # *ps* $\vdash q \Longrightarrow$ *ps* $\vdash p \longrightarrow q$›
  by (*metis imply.simps imply-append imply-swap-append*)

lemmas *deduct* [*iff*] = *deduct1 deduct2*

lemma *cut*: ‹p # *ps* $\vdash r \Longrightarrow q$ # *ps* $\vdash p \Longrightarrow q$ # *ps* $\vdash r$›
  by (*meson MP′ deduct*(*2*) *imply-Cons*)

lemma *Boole*: ‹($\neg$ p) # *ps* $\vdash \bot \Longrightarrow$ *ps* $\vdash p$›
  by (*meson MP′ Neg add-imply deduct*(*2*))

lemma *imply-weaken*: ‹*ps* $\vdash q \Longrightarrow$ *set ps* $\subseteq$ *set ps′* $\Longrightarrow$ *ps′* $\vdash q$›
  by (*induct ps arbitrary*: *q*) (*simp*, *metis MP′ deduct*(*2*) *imply-mem insert-subset list.simps*(*15*))

8 Consistent

definition ‹*consistent S* $\equiv \nexists S'$. *set* $S' \subseteq S \wedge S' \vdash \bot$›

lemma *UN-finite-bound*:
  assumes ‹*finite A*› **and** ‹$A \subseteq (\bigcup n.\ f\ n)$›
  shows ‹$\exists m$ :: *nat*. $A \subseteq (\bigcup n \leq m.\ f\ n)$›
  using *assms*
proof (*induct rule*: *finite-induct*)
  case (*insert x A*)
  then obtain m **where** ‹$A \subseteq (\bigcup n \leq m.\ f\ n)$›
    by *fast*
  then have ‹$A \subseteq (\bigcup n \leq (m + k).\ f\ n)$› **for** k
    by *fastforce*
  moreover obtain m' **where** ‹$x \in f\ m'$›
    using *insert*(*4*) **by** *blast*
  ultimately have ‹$\{x\} \cup A \subseteq (\bigcup n \leq m + m'.\ f\ n)$›

by *auto*
then show *?case*
by *blast*
qed *simp*

lemma *split-list*:
assumes ‹$x \in set\ A$›
shows ‹$set\ (x\ \#\ removeAll\ x\ A) = set\ A \wedge x \notin set\ (removeAll\ x\ A)$›
using *assms* **by** *auto*

lemma *imply-params-fm*: ‹$params\text{-}fm\ (ps \rightsquigarrow q) = params\text{-}fm\ q \cup (\bigcup p \in set\ ps.\ params\text{-}fm\ p)$›
by (*induct ps*) *auto*

lemma *inconsistent-fm*:
assumes ‹*consistent S*› **and** ‹$\neg$ *consistent* $(\{p\} \cup S)$›
obtains S' **where** ‹$set\ S' \subseteq S$› **and** ‹$p\ \#\ S' \vdash \bot$›
proof −
obtain S' **where** S': ‹$set\ S' \subseteq \{p\} \cup S$› ‹$p \in set\ S'$› ‹$S' \vdash \bot$›
using *assms* **unfolding** *consistent-def* **by** *blast*
then obtain S'' **where** S'': ‹$set\ (p\ \#\ S'') = set\ S'$› ‹$p \notin set\ S''$›
using *split-list* **by** *metis*
then have ‹$p\ \#\ S'' \vdash \bot$›
using ‹$S' \vdash \bot$› *imply-weaken* **by** *blast*
then show *?thesis*
using *that* S'' $S'(1)$ *Diff-insert-absorb Diff-subset-conv list.simps*(*15*) **by** *metis*
qed

lemma *consistent-add-witness*:
assumes ‹*consistent S*› **and** ‹$\neg\ (\forall\ p) \in S$› **and** ‹$a \notin params\ S$›
shows ‹*consistent* $(\{\neg\ \langle\star a/0\rangle p\} \cup S)$›
unfolding *consistent-def*
proof
assume ‹$\exists S'.\ set\ S' \subseteq \{\neg\ \langle\star a/0\rangle p\} \cup S \wedge S' \vdash \bot$›
then obtain S' **where** ‹$set\ S' \subseteq S$› **and** ‹$(\neg\ \langle\star a/0\rangle p)\ \#\ S' \vdash \bot$›
using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *metis*
then have ‹$\vdash \neg\ \langle\star a/0\rangle p \longrightarrow S' \rightsquigarrow \bot$›
by *simp*
moreover have ‹$a \notin params\text{-}fm\ p$›
using *assms*(*2*−*3*) **by** *auto*
moreover have ‹$\forall p \in set\ S'.\ a \notin params\text{-}fm\ p$›
using ‹$set\ S' \subseteq S$› *assms*(*3*) **by** *auto*
then have ‹$a \notin params\text{-}fm\ (S' \rightsquigarrow \bot)$›
by (*simp add*: *imply-params-fm*)
ultimately have ‹$\vdash \neg\ (\forall\ p) \longrightarrow S' \rightsquigarrow \bot$›
using GR' **by** *fast*
then have ‹$\neg\ (\forall\ p)\ \#\ S' \vdash \bot$›
by *simp*
moreover have ‹$set\ ((\neg\ (\forall\ p))\ \#\ S') \subseteq S$›

using ‹*set* $S' \subseteq S$› *assms*(*2*) **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

lemma *consistent-add-instance*:
assumes ‹*consistent* S› **and** ‹$\forall\, p \in S$›
shows ‹*consistent* $(\{\langle t/0\rangle p\} \cup S)$›
unfolding *consistent-def*
proof
assume ‹$\exists\, S'.$ *set* $S' \subseteq \{\langle t/0\rangle p\} \cup S \wedge S' \vdash \bot$›
then obtain S' **where** ‹*set* $S' \subseteq S$› **and** ‹$\langle t/0\rangle p \;\#\; S' \vdash \bot$›
using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *blast*
moreover have ‹$\vdash \forall\, p \longrightarrow \langle t/0\rangle p$›
using *IA* **by** *blast*
ultimately have ‹$\forall\, p \;\#\; S' \vdash \bot$›
by (*meson add-imply cut deduct*(*1*))
moreover have ‹*set* $((\forall\, p) \;\#\; S') \subseteq S$›
using ‹*set* $S' \subseteq S$› *assms*(*2*) **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

9 Extension

fun *witness* **where**
‹*witness used* $(\neg\, (\forall\, p)) = \{\neg\, \langle\star(\mathit{SOME}\ a.\ a \notin \mathit{used})/0\rangle p\}$›
| ‹*witness* - - $= \{\}$›

primrec *extend* **where**
‹*extend* $S\ f\ 0 = S$›
| ‹*extend* $S\ f\ (\mathit{Suc}\ n) =$
(*let* $Sn =$ *extend* $S\ f\ n$ *in*
if consistent $(\{f\ n\} \cup Sn)$
then witness (*params* $(\{f\ n\} \cup Sn))\ (f\ n) \cup \{f\ n\} \cup Sn$
else Sn)›

definition ‹*Extend* $S\ f \equiv \bigcup n.$ *extend* $S\ f\ n$›

lemma *extend-subset*: ‹$S \subseteq$ *extend* $S\ f\ n$›
by (*induct n*) (*fastforce simp*: *Let-def*)+

lemma *Extend-subset*: ‹$S \subseteq$ *Extend* $S\ f$›
unfolding *Extend-def* **by** (*metis Union-upper extend.simps*(*1*) *range-eqI*)

lemma *extend-bound*: ‹$(\bigcup n \leq m.$ *extend* $S\ f\ n) =$ *extend* $S\ f\ m$›
by (*induct m*) (*simp-all add*: *atMost-Suc Let-def*)

lemma *finite-params-witness* [*simp*]: ‹*finite* (*params* (*witness used p*))›

by (*induct used p rule*: *witness.induct*) *simp-all*

lemma *finite-params-extend* [*simp*]: ‹*finite* (*params* (*extend S f n*) − *params S*)›
by (*induct n*) (*simp-all add*: *Let-def Un-Diff*)

lemma *Set-Diff-Un*: ‹$X - (Y \cup Z) = X - Y - Z$›
by *blast*

lemma *infinite-params-extend*:
assumes ‹*infinite* (*UNIV* − *params S*)›
shows ‹*infinite* (*UNIV* − *params* (*extend S f n*))›
proof −
have ‹*finite* (*params* (*extend S f n*) − *params S*)›
by *simp*
then obtain *extra* **where** ‹*finite extra*› ‹*params* (*extend S f n*) = *extra* ∪ *params S*›
using *extend-subset* **by** *fast*
then have ‹*?thesis* = *infinite* (*UNIV* − (*extra* ∪ *params S*))›
by *simp*
also have ‹. . . = *infinite* (*UNIV* − *extra* − *params S*)›
by (*simp add*: *Set-Diff-Un*)
also have ‹. . . = *infinite* (*UNIV* − *params S*)›
using ‹*finite extra*› **by** (*metis Set-Diff-Un Un-commute finite-Diff2*)
finally show *?thesis*
using *assms* **..**
qed

lemma *consistent-witness*:
assumes ‹*consistent S*› **and** ‹$p \in S$› **and** ‹*params S* ⊆ *used*›
and ‹*infinite* (*UNIV* − *used*)›
shows ‹*consistent* (*witness used p* ∪ *S*)›
using *assms*
proof (*induct used p rule*: *witness.induct*)
case (*1 used p*)
moreover have ‹∃ *a*. *a* ∉ *used*›
using *1*(*4*) **by** (*meson Diff-iff finite-params-fm finite-subset subset-iff*)
ultimately obtain *a* **where** *a*: ‹*witness used* (¬ (∀ *p*)) = {¬ ⟨⋆*a*/*0*⟩*p*}› **and** ‹*a* ∉ *used*›
by (*metis someI-ex witness.simps*(*1*))
then have ‹*a* ∉ *params S*›
using *1*(*3*) **by** *fast*
then show *?case*
using *1*(*1*–*2*) *a*(*1*) *consistent-add-witness* **by** *metis*
qed (*auto simp*: *assms*)

lemma *consistent-extend*:
assumes ‹*consistent S*› **and** ‹*infinite* (*UNIV* − *params S*)›
shows ‹*consistent* (*extend S f n*)›
proof (*induct n*)

case (*Suc n*)
have ‹*infinite* (*UNIV* − *params* (*extend S f n*))›
using *assms*(*2*) *infinite-params-extend* **by** *fast*
with *finite-params-fm* **have** ‹*infinite* (*UNIV* − (*params-fm* (*f n*) ∪ *params* (*extend S f n*)))›
by (*metis Set-Diff-Un Un-commute finite-Diff2*)
with *Suc consistent-witness*[**where** *S*=‹{*f n*} ∪ *extend S f n*›] **show** *?case*
by (*simp add*: *Let-def*)
qed (*use assms*(*1*) **in** *simp*)

lemma *consistent-Extend*:
assumes ‹*consistent S*› **and** ‹*infinite* (*UNIV* − *params S*)›
shows ‹*consistent* (*Extend S f*)›
unfolding *consistent-def*
proof
assume ‹$\exists S'.$ *set* $S' \subseteq$ *Extend S f* $\wedge$ $S' \vdash \bot$›
then obtain S' **where** ‹$S' \vdash \bot$› **and** ‹*set* $S' \subseteq$ *Extend S f*›
unfolding *consistent-def* **by** *blast*
then obtain *m* **where** ‹*set* $S' \subseteq (\bigcup n \leq m.$ *extend S f n*)›
unfolding *Extend-def* **using** *UN-finite-bound* **by** (*metis finite-set*)
then have ‹*set* $S' \subseteq$ *extend S f m*›
using *extend-bound* **by** *blast*
moreover have ‹*consistent* (*extend S f m*)›
using *assms consistent-extend* **by** *blast*
ultimately show *False*
unfolding *consistent-def* **using** ‹$S' \vdash \bot$› **by** *blast*
qed

10 Maximal

definition ‹*maximal S* $\equiv \forall p.\ p \notin S \longrightarrow \neg$ *consistent* ($\{p\} \cup S$)›

lemma *maximal-exactly-one*:
assumes ‹*consistent S*› **and** ‹*maximal S*›
shows ‹$p \in S \longleftrightarrow (\neg\ p) \notin S$›
proof
assume ‹$p \in S$›
show ‹$(\neg\ p) \notin S$›
proof
assume ‹$(\neg\ p) \in S$›
then have ‹*set* $[p, \neg\ p] \subseteq S$›
using ‹$p \in S$› **by** *simp*
moreover have ‹$[p, \neg\ p] \vdash \bot$›
by *blast*
ultimately show *False*
using ‹*consistent S*› **unfolding** *consistent-def* **by** *blast*
qed
next
assume ‹$(\neg\ p) \notin S$›

```
  then have ‹¬ consistent ({¬ p} ∪ S)›
    using ‹maximal S› unfolding maximal-def by blast
  then obtain S' where ‹set S' ⊆ S› ‹(¬ p) # S' ⊢ ⊥›
    using ‹consistent S› inconsistent-fm by blast
  then have ‹S' ⊢ p›
    using Boole by blast
  have ‹consistent ({p} ∪ S)›
    unfolding consistent-def
  proof
    assume ‹∃ S'. set S' ⊆ {p} ∪ S ∧ S' ⊢ ⊥›
    then obtain S'' where ‹set S'' ⊆ S› and ‹p # S'' ⊢ ⊥›
      using assms inconsistent-fm unfolding consistent-def by blast
    then have ‹S' @ S'' ⊢ ⊥›
      using ‹S' ⊢ p› by (metis MP' add-imply imply.simps(2) imply-append)
    moreover have ‹set (S' @ S'') ⊆ S›
      using ‹set S' ⊆ S› ‹set S'' ⊆ S› by simp
    ultimately show False
      using ‹consistent S› unfolding consistent-def by blast
  qed
  then show ‹p ∈ S›
    using ‹maximal S› unfolding maximal-def by blast
qed

lemma maximal-Extend:
  assumes ‹surj f›
  shows ‹maximal (Extend S f)›
  unfolding maximal-def
proof safe
  fix p
  assume ‹p ∉ Extend S f› and ‹consistent ({p} ∪ Extend S f)›
  obtain k where k: ‹f k = p›
    using ‹surj f› unfolding surj-def by metis
  then have ‹p ∉ extend S f (Suc k)›
    using ‹p ∉ Extend S f› unfolding Extend-def by blast
  then have ‹¬ consistent ({p} ∪ extend S f k)›
    using k by (auto simp: Let-def)
  moreover have ‹{p} ∪ extend S f k ⊆ {p} ∪ Extend S f›
    unfolding Extend-def by blast
  ultimately have ‹¬ consistent ({p} ∪ Extend S f)›
    unfolding consistent-def by auto
  then show False
    using ‹consistent ({p} ∪ Extend S f)› by blast
qed
```

11 Saturation

```
definition ‹saturated S ≡ ∀ p. ¬ (∀ p) ∈ S ⟶ (∃ a. (¬ ⟨⋆a/0⟩p) ∈ S)›

lemma saturated-Extend:
```

assumes ‹*consistent* (*Extend S f*)› **and** ‹*surj f*›
shows ‹*saturated* (*Extend S f*)›
unfolding *saturated-def*
proof *safe*
fix *p*
assume ∗: ‹¬ (∀ *p*) ∈ *Extend S f*›
obtain *k* **where** *k*: ‹*f k* = ¬ (∀ *p*)›
using ‹*surj f*› **unfolding** *surj-def* **by** *metis*
have ‹*extend S f k* ⊆ *Extend S f*›
unfolding *Extend-def* **by** *auto*
then have ‹*consistent* ({¬ (∀ *p*)} ∪ *extend S f k*)›
using *assms*(*1*) ∗ **unfolding** *consistent-def* **by** *blast*
then have ‹∃ *a*. *extend S f* (*Suc k*) = {¬ ⟨⋆*a*/*0*⟩*p*} ∪ {¬ (∀ *p*)} ∪ *extend S f k*›
using *k* **by** (*auto simp*: *Let-def*)
moreover have ‹*extend S f* (*Suc k*) ⊆ *Extend S f*›
unfolding *Extend-def* **by** *blast*
ultimately show ‹∃ *a*. ¬ ⟨⋆ *a*/*0*⟩*p* ∈ *Extend S f*›
by *blast*
qed

12 Hintikka

locale *Hintikka* =
fixes *H* :: ‹(′*f*, ′*p*) *fm set*›
assumes
FlsH: ‹⊥ ∉ *H*› **and**
ImpH: ‹(*p* ⟶ *q*) ∈ *H* ⟷ (*p* ∈ *H* ⟶ *q* ∈ *H*)› **and**
UniH: ‹(∀ *p* ∈ *H*) ⟷ (∀ *t*. ⟨*t*/*0*⟩*p* ∈ *H*)›

12.1 Model Existence

abbreviation *hmodel* (‹⟦-⟧›) **where** ‹⟦*H*⟧ ≡ ⟦#, †, λ*P ts*. ‡*P ts* ∈ *H*⟧›

lemma *semantics-tm-id* [*simp*]: ‹⦇#, †⦈ *t* = *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *semantics-tm-id-map* [*simp*]: ‹*map* ⦇#, †⦈ *ts* = *ts*›
by (*auto cong*: *map-cong*)

theorem *Hintikka-model*:
assumes ‹*Hintikka H*›
shows ‹*p* ∈ *H* ⟷ ⟦*H*⟧ *p*›
proof (*induct p rule*: *wf-induct*[**where** *r*=‹*measure size-fm*›])
case *1*
then show *?case* **..**
next
case (*2 x*)
then show *?case*
using *assms* **unfolding** *Hintikka-def* **by** (*cases x*) *auto*

qed

12.2 Maximal Consistent Sets are Hintikka Sets

lemma *deriv-iff-MCS*:
  assumes ‹*consistent* S› **and** ‹*maximal* S›
  shows ‹$(\exists\, ps.\ set\ ps \subseteq S \wedge ps \vdash p) \longleftrightarrow p \in S$›
proof
  from *assms maximal-exactly-one*[*OF assms*(*1*)] **show** ‹$\exists\, ps.\ set\ ps \subseteq S \wedge ps \vdash p \Longrightarrow p \in S$›
    unfolding *consistent-def* **using** *MP add-imply deduct1 imply.simps*(*1*) *imply-ImpE*
    by (*metis insert-absorb insert-mono list.simps*(*15*))
next
  show ‹$p \in S \Longrightarrow \exists\, ps.\ set\ ps \subseteq S \wedge ps \vdash p$›
    using *imply-head* **by** (*metis empty-subsetI insert-absorb insert-mono list.set*(*1*) *list.simps*(*15*))
qed

lemma *Hintikka-Extend*:
  assumes ‹*consistent* H› **and** ‹*maximal* H› **and** ‹*saturated* H›
  shows ‹*Hintikka* H›
proof
  show ‹$\bot \notin H$›
    using *assms deriv-iff-MCS* **unfolding** *consistent-def* **by** *fast*
next
  fix $p\ q$
  show ‹$(p \longrightarrow q) \in H \longleftrightarrow (p \in H \longrightarrow q \in H)$›
    using *deriv-iff-MCS*[*OF assms*(*1–2*)] *maximal-exactly-one*[*OF assms*(*1–2*)]
    by (*metis Imp1 contraposition add-imply deduct1 insert-subset list.simps*(*15*))
next
  fix p
  show ‹$(\forall\, p \in H) \longleftrightarrow (\forall\, t.\ \langle t/0\rangle p \in H)$›
    using *assms consistent-add-instance maximal-exactly-one*
    unfolding *maximal-def saturated-def* **by** *metis*
qed

13 Countable Formulas

instance *tm* :: (*countable*) *countable*
  by *countable-datatype*

instance *fm* :: (*countable*, *countable*) *countable*
  by *countable-datatype*

14 Completeness

lemma *infinite-Diff-fin-Un*: ‹*infinite* $(X - Y) \Longrightarrow$ *finite* $Z \Longrightarrow$ *infinite* $(X - (Z \cup Y))$›
  by (*simp add*: *Set-Diff-Un Un-commute*)

theorem *strong-completeness*:
  fixes *p* :: ‹(*'f* :: *countable*, *'p* :: *countable*) *fm*›
  assumes ‹$\forall$ (*E* :: - $\Rightarrow$ *'f tm*) *F G*. ($\forall q \in X$. $[\![E, F, G]\!]$ *q*) $\longrightarrow$ $[\![E, F, G]\!]$ *p*›
    and ‹*infinite* (*UNIV* $-$ *params X*)›
  shows ‹$\exists$ *ps*. *set ps* $\subseteq X \wedge$ *ps* $\vdash$ *p*›
proof (*rule ccontr*)
  assume ‹$\nexists$ *ps*. *set ps* $\subseteq X \wedge$ *ps* $\vdash$ *p*›
  then have $*$: ‹$\nexists$ *ps*. *set ps* $\subseteq X \wedge$ (($\neg$ *p*) # *ps* $\vdash \bot$)›
    using *Boole* **by** *blast*

  let *?S* = ‹$\{\neg$ *p*$\} \cup X$›
  let *?H* = ‹*Extend ?S from-nat*›

  from *inconsistent-fm* **have** ‹*consistent ?S*›
    unfolding *consistent-def* **using** $*$ *imply-Cons* **by** *metis*
  moreover have ‹*infinite* (*UNIV* $-$ *params ?S*)›
    using *assms*(*2*) *finite-params-fm* **by** (*simp add*: *infinite-Diff-fin-Un*)
  ultimately have ‹*consistent ?H*› **and** ‹*maximal ?H*›
    using *consistent-Extend maximal-Extend surj-from-nat* **by** *blast+*
  moreover from *this* **have** ‹*saturated ?H*›
    using *saturated-Extend* **by** *fastforce*
  ultimately have ‹*Hintikka ?H*›
    using *assms*(*2*) *Hintikka-Extend* **by** *blast*

  have ‹$[\![$*?H*$]\!]$ *p*› **if** ‹*p* $\in$ *?S*› **for** *p*
    using *that Extend-subset Hintikka-model* ‹*Hintikka ?H*› **by** *blast*
  then have ‹$[\![$*?H*$]\!]$ ($\neg$ *p*)› **and** ‹$\forall q \in X$. $[\![$*?H*$]\!]$ *q*›
    by *blast+*
  moreover from *this* **have** ‹$[\![$*?H*$]\!]$ *p*›
    using *assms*(*1*) **by** *blast*
  ultimately show *False*
    by *simp*
qed

theorem *completeness*:
  fixes *p* :: ‹(*nat*, *nat*) *fm*›
  assumes ‹$\forall$ (*E* :: *nat* $\Rightarrow$ *nat tm*) *F G*. $[\![E, F, G]\!]$ *p*›
  shows ‹$\vdash$ *p*›
  using *assms strong-completeness*[**where** *X*=‹{}›] **by** *auto*

15 Main Result

abbreviation *valid* :: ‹(*nat*, *nat*) *fm* $\Rightarrow$ *bool*› **where**

‹valid p ≡ ∀ (E :: nat ⇒ nat tm) F G. ⟦E, F, G⟧ p›

theorem *main*: ‹valid p ⟷ (⊢ p)›
using *completeness soundness* **by** *blast*

end

theory *FOL-Axiomatic-Variant* **imports** *HOL−Library.Countable* **begin**

16 Syntax

datatype *'f tm*
= *Var nat* (‹#›)
| *Fun 'f* ‹*'f tm list*› (‹†›)

datatype (*'f*, *'p*) *fm*
= *Falsity* (‹⊥›)
| *Pre 'p* ‹*'f tm list*› (‹‡›)
| *Imp* ‹(*'f*, *'p*) *fm*› ‹(*'f*, *'p*) *fm*› (**infixr** ‹⟶› *55*)
| *Uni* ‹(*'f*, *'p*) *fm*› (‹∀›)

abbreviation *Neg* (‹¬ -› [*70*] *70*) **where** ‹¬ p ≡ p ⟶ ⊥›

term ‹∀ (⊥ ⟶ ‡''P'' [†''f'' [#0]])›

17 Semantics

definition *shift* :: ‹(nat ⇒ 'a) ⇒ nat ⇒ 'a ⇒ nat ⇒ 'a›
(‹-⟨-:-⟩› [*90*, *0*, *0*] *91*) **where**
‹E⟨n:x⟩ = (λm. if m < n then E m else if m = n then x else E (m−1))›

primrec *semantics-tm* (‹⦇-, -⦈›) **where**
‹⦇E, F⦈ (#n) = E n›
| ‹⦇E, F⦈ (†f ts) = F f (map ⦇E, F⦈ ts)›

primrec *semantics-fm* (‹⟦-, -, -⟧›) **where**
‹⟦-, -, -⟧ ⊥ = False›
| ‹⟦E, F, G⟧ (‡P ts) = G P (map ⦇E, F⦈ ts)›
| ‹⟦E, F, G⟧ (p ⟶ q) = (⟦E, F, G⟧ p ⟶ ⟦E, F, G⟧ q)›
| ‹⟦E, F, G⟧ (∀ p) = (∀ x. ⟦E⟨0:x⟩, F, G⟧ p)›

proposition ‹⟦E, F, G⟧ (∀ (‡P [# 0]) ⟶ ‡P [† a []])›
by (*simp add*: *shift-def*)

18 Operations

18.1 Shift

lemma *shift-eq* [*simp*]: ‹$n = m \Longrightarrow (E\langle n{:}x\rangle)\ m = x$›
  by (*simp add*: *shift-def*)

lemma *shift-gt* [*simp*]: ‹$m < n \Longrightarrow (E\langle n{:}x\rangle)\ m = E\ m$›
  by (*simp add*: *shift-def*)

lemma *shift-lt* [*simp*]: ‹$n < m \Longrightarrow (E\langle n{:}x\rangle)\ m = E\ (m-1)$›
  by (*simp add*: *shift-def*)

lemma *shift-commute* [*simp*]: ‹$E\langle n{:}y\rangle\langle 0{:}x\rangle = E\langle 0{:}x\rangle\langle n+1{:}y\rangle$›
proof
  fix *m*
  show ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle)\ m = (E\langle 0{:}x\rangle\langle n+1{:}y\rangle)\ m$›
    unfolding *shift-def* **by** (*cases m*) *simp-all*
qed

18.2 Variables

primrec *vars-tm* **where**
  ‹*vars-tm* (#*n*) = [*n*]›
| ‹*vars-tm* (†- *ts*) = *concat* (*map vars-tm ts*)›

primrec *vars-fm* **where**
  ‹*vars-fm* ⊥ = []›
| ‹*vars-fm* (‡- *ts*) = *concat* (*map vars-tm ts*)›
| ‹*vars-fm* (*p* ⟶ *q*) = *vars-fm p* @ *vars-fm q*›
| ‹*vars-fm* (∀ *p*) = *vars-fm p*›

abbreviation ‹*vars S* ≡ ⋃ *p* ∈ *S*. *set* (*vars-fm p*)›

primrec *max-list* :: ‹*nat list* ⇒ *nat*› **where**
  ‹*max-list* [] = *0*›
| ‹*max-list* (*x* # *xs*) = *max x* (*max-list xs*)›

lemma *max-list-append*: ‹*max-list* (*xs* @ *ys*) = *max* (*max-list xs*) (*max-list ys*)›
  by (*induct xs*) *auto*

lemma *upd-vars-tm* [*simp*]: ‹$n \notin set\ (vars\text{-}tm\ t) \Longrightarrow (\!|E(n := x),\ F|\!)\ t = (\!|E,\ F|\!)\ t$›
  by (*induct t*) (*auto cong*: *map-cong*)

lemma *shift-upd-commute*: ‹$m \le n \Longrightarrow (E(n := x)\langle m{:}y\rangle) = ((E\langle m{:}y\rangle)(n + 1 := x))$›
  unfolding *shift-def* **by** *fastforce*

lemma *max-list-concat*: ‹$xs \in set\ xss \Longrightarrow max\text{-}list\ xs \le max\text{-}list\ (concat\ xss)$›

by (*induct xss*) (*auto simp*: *max-list-append*)

lemma *max-list-in*: ‹*max-list xs* < *n* ⟹ *n* ∉ *set xs*›
  by (*induct xs*) *auto*

lemma *upd-vars-fm* [*simp*]: ‹*max-list* (*vars-fm p*) < *n* ⟹ ⟦*E*(*n* := *x*), *F*, *G*⟧ *p* = ⟦*E*, *F*, *G*⟧ *p*›
proof (*induct p arbitrary*: *E n*)
  case (*Pre P ts*)
  moreover have ‹*max-list* (*concat* (*map vars-tm ts*)) < *n*›
    using *Pre.prems max-list-concat* **by** *simp*
  then have ‹*n* ∉ *set* (*concat* (*map vars-tm ts*))›
    using *max-list-in* **by** *blast*
  then have ‹∀ *t* ∈ *set ts*. *n* ∉ *set* (*vars-tm t*)›
    by *simp*
  ultimately show *?case*
    using *upd-vars-tm* **by** (*metis list.map-cong semantics-fm.simps*(*2*))
next
  case (*Uni p*)
  have ‹*?case* = ((∀ *y*. ⟦*E*(*n* := *x*)⟨*0*:*y*⟩, *F*, *G*⟧ *p*) = (∀ *y*. ⟦*E*⟨*0*:*y*⟩, *F*, *G*⟧ *p*))›
    by (*simp add*: *fun-upd-def*)
  also have ‹... = ((∀ *y*. ⟦(*E*⟨*0*:*y*⟩)(*n* + *1* := *x*), *F*, *G*⟧ *p*) = (∀ *y*. ⟦*E*⟨*0*:*y*⟩, *F*, *G*⟧ *p*))›
    by (*simp add*: *shift-upd-commute*)
  finally show *?case*
    using *Uni* **by** *fastforce*
qed (*auto simp*: *max-list-append cong*: *map-cong*)

abbreviation ‹*max-var p* ≡ *max-list* (*vars-fm p*)›

18.3 Instantiation

primrec *lift-tm* (‹↑›) **where**
  ‹↑(#*n*) = #(*n*+*1*)›
| ‹↑(†*f ts*) = †*f* (*map* ↑ *ts*)›

primrec *inst-tm* (‹-'⟪-'/-'⟫› [*90*, *0*, *0*] *91*) **where**
  ‹(#*n*)⟪*s*/*m*⟫ = (*if n* < *m then* #*n else if n* = *m then s else* #(*n*−*1*))›
| ‹(†*f ts*)⟪*s*/*m*⟫ = †*f* (*map* (λ*t*. *t*⟪*s*/*m*⟫) *ts*)›

primrec *inst-fm* (‹-'⟨-'/-'⟩› [*90*, *0*, *0*] *91*) **where**
  ‹⊥⟨-/-⟩ = ⊥›
| ‹(‡*P ts*)⟨*s*/*m*⟩ = ‡*P* (*map* (λ*t*. *t*⟪*s*/*m*⟫) *ts*)›
| ‹(*p* ⟶ *q*)⟨*s*/*m*⟩ = (*p*⟨*s*/*m*⟩ ⟶ *q*⟨*s*/*m*⟩)›
| ‹(∀ *p*)⟨*s*/*m*⟩ = ∀ (*p*⟨↑*s*/*m*+*1*⟩)›

lemma *lift-lemma* [*simp*]: ‹⦇*E*⟨*0*:*x*⟩, *F*⦈ (↑*t*) = ⦇*E*, *F*⦈ *t*›
  by (*induct t*) (*auto cong*: *map-cong*)

lemma *inst-tm-semantics* [*simp*]: ‹$(\!|E,\ F|\!)\ (t\langle\!\langle s/m\rangle\!\rangle) = (\!|E\langle m{:}(\!|E,\ F|\!)\ s\rangle,\ F|\!)\ t$›
  by (*induct t*) (*auto cong*: *map-cong*)

lemma *inst-fm-semantics* [*simp*]: ‹$[\![E,\ F,\ G]\!]\ (p\langle t/m\rangle) = [\![E\langle m{:}(\!|E,\ F|\!)\ t\rangle,\ F,\ G]\!]$ p›
  by (*induct p arbitrary*: *E m t*) (*auto cong*: *map-cong*)

18.4 Size

The built-in *size* is not invariant under substitution.

primrec *size-fm* **where**
  ‹*size-fm* $\bot$ = *1*›
| ‹*size-fm* ($\ddagger$- -) = *1*›
| ‹*size-fm* ($p \longrightarrow q$) = *1* + *size-fm p* + *size-fm q*›
| ‹*size-fm* ($\forall\, p$) = *1* + *size-fm p*›

lemma *size-inst-fm* [*simp*]:
  ‹*size-fm* ($p\langle t/m\rangle$) = *size-fm p*›
  by (*induct p arbitrary*: *m t*) *auto*

19 Propositional Semantics

primrec *boolean* **where**
  ‹*boolean* - - $\bot$ = *False*›
| ‹*boolean G* - ($\ddagger P\ ts$) = *G P ts*›
| ‹*boolean G A* ($p \longrightarrow q$) = (*boolean G A p* $\longrightarrow$ *boolean G A q*)›
| ‹*boolean* - *A* ($\forall\, p$) = *A* ($\forall\, p$)›

abbreviation ‹*tautology p* $\equiv \forall$ *G A*. *boolean G A p*›

proposition ‹*tautology* ($\forall$ ($\ddagger P$ [#*0*]) $\longrightarrow \forall$ ($\ddagger P$ [#*0*]))›
  by *simp*

lemma *boolean-semantics*: ‹*boolean* (λa. *G a* $\circ$ *map* $(\!|E,\ F|\!)$) $[\![E,\ F,\ G]\!] = [\![E,\ F,$ $G]\!]$›
proof
  fix *p*
  show ‹*boolean* (λa. *G a* $\circ$ *map* $(\!|E,\ F|\!)$) $[\![E,\ F,\ G]\!]$ *p* = $[\![E,\ F,\ G]\!]$ *p*›
    by (*induct p*) *simp-all*
qed

lemma *tautology*: ‹*tautology p* $\Longrightarrow [\![E,\ F,\ G]\!]$ *p*›
  using *boolean-semantics* **by** *metis*

proposition ‹$\exists\, p$. ($\forall$ *E F G*. $[\![E,\ F,\ G]\!]$ *p*) $\wedge\ \neg$ *tautology p*›
  by (*metis boolean.simps*(*4*) *fm.simps*(*36*) *semantics-fm.simps*(*1*,*3*,*4*))

20 Calculus

Adapted from System Q1 by Smullyan in First-Order Logic (1968)

inductive *Axiomatic* (‹⊢ -› [*50*] *50*) **where**
  TA: ‹*tautology* $p \Longrightarrow \vdash p$›
| *IA*: ‹$\vdash \forall\, p \longrightarrow p\langle t/0\rangle$›
| *MP*: ‹$\vdash p \longrightarrow q \Longrightarrow \vdash p \Longrightarrow \vdash q$›
| *GR*: ‹$\vdash q \longrightarrow p\langle \#n/0\rangle \Longrightarrow$ *max-var* $p < n \Longrightarrow$ *max-var* $q < n \Longrightarrow \vdash q \longrightarrow \forall\, p$›

lemmas
  TA[*simp*]
  MP[*trans*, *dest*]
  GR[*intro*]

We simulate assumptions on the lhs of ⊢ with a chain of implications on the rhs.

primrec *imply* (**infixr** ‹⇝› *56*) **where**
  ‹$([] \rightsquigarrow q) = q$›
| ‹$(p \# ps \rightsquigarrow q) = (p \longrightarrow ps \rightsquigarrow q)$›

abbreviation *Axiomatic-assms* (‹- ⊢ -› [*50*, *50*] *50*) **where**
  ‹$ps \vdash q \equiv \vdash ps \rightsquigarrow q$›

21 Soundness

theorem *soundness*: ‹$\vdash p \Longrightarrow [\![E,\ F,\ G]\!]\ p$›
proof (*induct p arbitrary*: *E F rule*: *Axiomatic.induct*)
  case (*GR q p n*)
  then have ‹$[\![E(n := x),\ F,\ G]\!]\ (q \longrightarrow p\langle \#n/0\rangle)$› **for** x
    by *blast*
  then have ‹$[\![E(n := x),\ F,\ G]\!]\ q \longrightarrow [\![E(n := x),\ F,\ G]\!]\ (p\langle \#n/0\rangle)$› **for** x
    by *simp*
  then have ‹$[\![E,\ F,\ G]\!]\ q \longrightarrow [\![E(n := x),\ F,\ G]\!]\ (p\langle \#n/0\rangle)$› **for** x
    using *GR.hyps*(*3*−*4*) **by** *simp*
  then have ‹$[\![E,\ F,\ G]\!]\ q \longrightarrow (\forall\, x.\ [\![E(n := x),\ F,\ G]\!]\ (p\langle \#n/0\rangle))$›
    by *blast*
  then have ‹$[\![E,\ F,\ G]\!]\ q \longrightarrow (\forall\, x.\ [\![E(n := x)\langle 0{:}x\rangle,\ F,\ G]\!]\ p)$›
    by *simp*
  then have ‹$[\![E,\ F,\ G]\!]\ q \longrightarrow (\forall\, x.\ [\![(E\langle 0{:}x\rangle)(n + 1 := x),\ F,\ G]\!]\ p)$›
    using *shift-upd-commute* **by** (*metis zero-le*)
  moreover have ‹*max-list* (*vars-fm* p) $< n$›
    using *GR.hyps*(*3*) **by** (*simp add*: *max-list-append*)
  ultimately have ‹$[\![E,\ F,\ G]\!]\ q \longrightarrow (\forall\, x.\ [\![E\langle 0{:}x\rangle,\ F,\ G]\!]\ p)$›
    using *upd-vars-fm* **by** *simp*
  then show *?case*
    by *simp*
qed (*auto simp*: *tautology*)

corollary ‹¬ (⊢ ⊥)›
  using *soundness* **by** *fastforce*

22 Derived Rules

lemma *AS*: ‹⊢ $(p \longrightarrow q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$›
  by *auto*

lemma *AK*: ‹⊢ $q \longrightarrow p \longrightarrow q$›
  by *auto*

lemma *Neg*: ‹⊢ $\neg\ \neg\ p \longrightarrow p$›
  by *auto*

lemma *contraposition*:
  ‹⊢ $(p \longrightarrow q) \longrightarrow \neg\ q \longrightarrow \neg\ p$›
  ‹⊢ $(\neg\ q \longrightarrow \neg\ p) \longrightarrow p \longrightarrow q$›
  by (*auto intro*: *TA*)

lemma *GR′*: ‹⊢ $\neg\ p\langle\#n/0\rangle \longrightarrow q \Longrightarrow$ *max-var* $p < n \Longrightarrow$ *max-var* $q < n \Longrightarrow \vdash$ $\neg\ \forall\, p \longrightarrow q$›
proof −
  assume ∗: ‹⊢ $\neg\ p\langle\#n/0\rangle \longrightarrow q$› **and** *n*: ‹*max-var* $p < n$› ‹*max-var* $q < n$›
  have ‹⊢ $\neg\ q \longrightarrow \neg\ \neg\ p\langle\#n/0\rangle$›
    using ∗ *contraposition*(*1*) **by** *fast*
  then have ‹⊢ $\neg\ q \longrightarrow p\langle\#n/0\rangle$›
    by (*meson AK AS MP Neg*)
  then have ‹⊢ $\neg\ q \longrightarrow \forall\, p$›
    using *n* **by** *auto*
  then have ‹⊢ $\neg\ \forall\, p \longrightarrow \neg\ \neg\ q$›
    using *contraposition*(*1*) **by** *fast*
  then show *?thesis*
    by (*meson AK AS MP Neg*)
qed

lemma *Imp3*: ‹⊢ $(p \longrightarrow q \longrightarrow r) \longrightarrow ((s \longrightarrow p) \longrightarrow (s \longrightarrow q) \longrightarrow s \longrightarrow r)$›
  by *auto*

lemma *imply-ImpE*: ‹⊢ $ps \leadsto p \longrightarrow ps \leadsto (p \longrightarrow q) \longrightarrow ps \leadsto q$›
  by (*induct ps*) (*auto intro*: *Imp3 MP*)

lemma *MP′* [*trans*, *dest*]: ‹$ps \vdash p \longrightarrow q \Longrightarrow ps \vdash p \Longrightarrow ps \vdash q$›
  using *imply-ImpE* **by** *fast*

lemma *imply-Cons* [*intro*]: ‹$ps \vdash q \Longrightarrow p\ \#\ ps \vdash q$›
  by (*auto intro*: *MP AK*)

lemma *imply-head* [*intro*]: ‹$p\ \#\ ps \vdash p$›
proof (*induct ps*)

case (*Cons q ps*)
then show *?case*
by (*metis AK MP′ imply.simps*(*1*−*2*))
qed *auto*

lemma *imply-lift-Imp* [*simp*]:
assumes ‹$\vdash p \longrightarrow q$›
shows ‹$\vdash p \longrightarrow ps \rightsquigarrow q$›
using *assms MP MP′ imply-head* **by** (*metis imply.simps*(*2*))

lemma *add-imply* [*simp*]: ‹$\vdash q \Longrightarrow ps \vdash q$›
using *MP imply-head* **by** (*auto simp del*: *TA*)

lemma *imply-mem* [*simp*]: ‹$p \in set\ ps \Longrightarrow ps \vdash p$›
proof (*induct ps*)
case (*Cons q ps*)
then show *?case*
by (*metis imply-Cons imply-head set-ConsD*)
qed *simp*

lemma *deduct1*: ‹$ps \vdash p \longrightarrow q \Longrightarrow p\ \#\ ps \vdash q$›
by (*meson MP′ imply-Cons imply-head*)

lemma *imply-append* [*iff*]: ‹$(ps\ @\ qs \rightsquigarrow r) = (ps \rightsquigarrow qs \rightsquigarrow r)$›
by (*induct ps*) *simp-all*

lemma *imply-swap-append*: ‹$ps\ @\ qs \vdash r \Longrightarrow qs\ @\ ps \vdash r$›
proof (*induct qs arbitrary*: *ps*)
case (*Cons q qs*)
then show *?case*
by (*metis deduct1 imply.simps*(*2*) *imply-append*)
qed *simp*

lemma *deduct2*: ‹$p\ \#\ ps \vdash q \Longrightarrow ps \vdash p \longrightarrow q$›
by (*metis imply.simps*(*1*−*2*) *imply-append imply-swap-append*)

lemmas *deduct* [*iff*] = *deduct1 deduct2*

lemma *cut* [*trans*, *dest*]: ‹$p\ \#\ ps \vdash r \Longrightarrow q\ \#\ ps \vdash p \Longrightarrow q\ \#\ ps \vdash r$›
by (*meson MP′ deduct*(*2*) *imply-Cons*)

lemma *Boole*: ‹$(\neg\ p)\ \#\ ps \vdash \bot \Longrightarrow ps \vdash p$›
by (*meson MP′ Neg add-imply deduct*(*2*))

lemma *imply-weaken*: ‹$ps \vdash q \Longrightarrow set\ ps \subseteq set\ ps' \Longrightarrow ps' \vdash q$›
proof (*induct ps arbitrary*: *q*)
case (*Cons p ps*)
then show *?case*
by (*metis MP′ deduct*(*2*) *imply-mem insert-subset list.simps*(*15*))

qed *simp*

23 Consistent

definition ‹*consistent* $S \equiv \nexists S'.$ *set* $S' \subseteq S \wedge S' \vdash \bot$›

lemma *UN-finite-bound*:
  assumes ‹*finite* A› **and** ‹$A \subseteq (\bigcup n.\ f\ n)$›
  shows ‹$\exists\, m :: nat.\ A \subseteq (\bigcup n \leq m.\ f\ n)$›
  using *assms*
proof (*induct rule*: *finite-induct*)
  case (*insert x A*)
  then obtain m **where** ‹$A \subseteq (\bigcup n \leq m.\ f\ n)$›
    by *fast*
  then have ‹$A \subseteq (\bigcup n \leq (m + k).\ f\ n)$› **for** k
    by *fastforce*
  moreover obtain m' **where** ‹$x \in f\ m'$›
    using *insert*(*4*) **by** *blast*
  ultimately have ‹$\{x\} \cup A \subseteq (\bigcup n \leq m + m'.\ f\ n)$›
    by *auto*
  then show *?case*
    by *blast*
qed *simp*

lemma *split-list*:
  assumes ‹$x \in$ *set* A›
  shows ‹*set* (x # *removeAll* x A) = *set* $A \wedge x \notin$ *set* (*removeAll* x A)›
  using *assms* **by** *auto*

lemma *imply-vars-fm*: ‹*vars-fm* (*ps* $\rightsquigarrow$ *q*) = *concat* (*map vars-fm ps*) @ *vars-fm*
q›
  by (*induct ps*) *auto*

lemma *inconsistent-fm*:
  assumes ‹*consistent* S› **and** ‹$\neg$ *consistent* ($\{p\} \cup S$)›
  obtains S' **where** ‹*set* $S' \subseteq S$› **and** ‹p # $S' \vdash \bot$›
proof −
  obtain S' **where** S': ‹*set* $S' \subseteq \{p\} \cup S$› ‹$p \in$ *set* S'› ‹$S' \vdash \bot$›
    using *assms* **unfolding** *consistent-def* **by** *blast*
  then obtain S'' **where** S'': ‹*set* (p # S'') = *set* S'› ‹$p \notin$ *set* S''›
    using *split-list* **by** *metis*
  then have ‹p # $S'' \vdash \bot$›
    using ‹$S' \vdash \bot$› *imply-weaken* **by** *blast*
  then show *?thesis*
    using *that* S'' S'(*1*)
    by (*metis Diff-insert-absorb Diff-subset-conv list.simps*(*15*))
qed

definition *max-set* :: ‹*nat set* $\Rightarrow$ *nat*› **where**

‹*max-set* $X \equiv$ *if* $X = \{\}$ *then 0 else Max* X›

lemma *max-list-in-Cons*: ‹$xs \neq [] \Longrightarrow$ *max-list xs* $\in$ *set xs*›
proof (*induct xs*)
  case *Nil*
  then show *?case*
    by *simp*
next
  case (*Cons x xs*)
  then show *?case*
    by (*metis linorder-not-less list.set-intros*(*1*−*2*) *max.absorb2 max.absorb3*
      max-list.simps(*1*−*2*) *max-nat.right-neutral*)
qed

lemma *max-list-max*: ‹$\forall x \in$ *set xs*. $x \leq$ *max-list xs*›
  by (*induct xs*) *auto*

lemma *max-list-in-set*: ‹*finite* $S \Longrightarrow$ *set xs* $\subseteq S \Longrightarrow$ *max-list xs* $\leq$ *max-set* S›
  unfolding *max-set-def* **using** *max-list-in-Cons*
  by (*metis* (*mono-tags*, *lifting*) *Max-ge bot.extremum-uniqueI bot-nat-0.extremum*
max-list.simps(*1*)
    set-empty subsetD)

lemma *consistent-add-witness*:
  assumes ‹*consistent* S› **and** ‹$(\neg \forall p) \in S$›
    and ‹*finite* (*vars* S)› **and** ‹*max-set* (*vars* S) $< n$›
  shows ‹*consistent* $(\{\neg\ p\langle\#n/0\rangle\} \cup S)$›
  unfolding *consistent-def*
proof
  assume ‹$\exists S'$. *set* $S' \subseteq \{\neg\ p\langle\#n/0\rangle\} \cup S \wedge S' \vdash \bot$›
  then obtain S' **where** ‹*set* $S' \subseteq S$› **and** ‹$(\neg\ p\langle\#n/0\rangle)\ \#\ S' \vdash \bot$›
    using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *metis*
  then have ‹$\vdash \neg\ p\langle\#n/0\rangle \longrightarrow S' \leadsto \bot$›
    by *simp*
  moreover have ‹*max-list* (*vars-fm p*) $< n$›
    using *assms*(*2*−*4*) *max-list-in-set* **by** *fastforce*
  moreover have ‹$\forall p \in$ *set* S'. *max-list* (*vars-fm p*) $< n$›
    using ‹*set* $S' \subseteq S$› *assms*(*3*−*4*) *max-list-in-set*
    by (*meson Union-upper image-eqI order-le-less-trans subsetD*)
  then have ‹*max-list* (*concat* (*map vars-fm* S')) $< n$›
    using *assms*(*4*) **by** (*induct* S') (*auto simp*: *max-list-append*)
  then have ‹*max-list* (*vars-fm* $(S' \leadsto \bot)$) $< n$›
    unfolding *imply-vars-fm max-list-append* **by** *simp*
  ultimately have ‹$\vdash \neg \forall p \longrightarrow S' \leadsto \bot$›
    using GR' **unfolding** *max-list-append* **by** *auto*
  then have ‹$(\neg \forall p)\ \#\ S' \vdash \bot$›
    by *simp*
  moreover have ‹*set* $((\neg \forall p)\ \#\ S') \subseteq S$›
    using ‹*set* $S' \subseteq S$› *assms*(*2*) **by** *simp*

ultimately show *False*
  using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

lemma *consistent-add-instance*:
  assumes ‹*consistent S*› **and** ‹$\forall\, p \in S$›
  shows ‹*consistent* ($\{p\langle t/0\rangle\} \cup S$)›
  unfolding *consistent-def*
proof
  assume ‹$\exists\, S'.\ set\ S' \subseteq \{p\langle t/0\rangle\} \cup S \wedge S' \vdash \bot$›
  then obtain S' **where** ‹$set\ S' \subseteq S$› **and** ‹$p\langle t/0\rangle \ \#\ S' \vdash \bot$›
    using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *blast*
  moreover have ‹$\vdash \forall\, p \longrightarrow p\langle t/0\rangle$›
    using *IA* **by** *blast*
  ultimately have ‹$\forall\, p\ \#\ S' \vdash \bot$›
    by (*meson add-imply cut deduct*(*1*))
  moreover have ‹$set\ ((\forall\, p)\ \#\ S') \subseteq S$›
    using ‹$set\ S' \subseteq S$› *assms*(*2*) **by** *simp*
  ultimately show *False*
    using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

24 Extension

fun *witness* **where**
  ‹$witness\ used\ (\neg\ \forall\, p) = \{\neg\ p\langle\#(SOME\ n.\ max\text{-}set\ used < n)/0\rangle\}$›
| ‹*witness - -* = {}›

primrec *extend* **where**
  ‹*extend S f 0 = S*›
| ‹*extend S f* (*Suc n*) =
   (*let Sn = extend S f n in*
     if consistent ({*f n*} ∪ *Sn*)
     then witness (*vars* ({*f n*} ∪ *Sn*)) (*f n*) ∪ {*f n*} ∪ *Sn*
     else Sn)›

definition ‹$Extend\ S\ f \equiv \bigcup n.\ extend\ S\ f\ n$›

lemma *Extend-subset*: ‹$S \subseteq Extend\ S\ f$›
  unfolding *Extend-def* **by** (*metis Union-upper extend.simps*(*1*) *range-eqI*)

lemma *extend-bound*: ‹$(\bigcup n \le m.\ extend\ S\ f\ n) = extend\ S\ f\ m$›
  by (*induct m*) (*simp-all add*: *atMost-Suc Let-def*)

lemma *finite-vars-witness* [*simp*]: ‹*finite* (*vars* (*witness used p*))›
  by (*induct used p rule*: *witness.induct*) *simp-all*

lemma *finite-vars-extend* [*simp*]: ‹*finite* (*vars S*) ⟹ *finite* (*vars* (*extend S f n*))›
  by (*induct n*) (*simp-all add*: *Let-def*)

```
lemma max-list-mono: ‹set xs ⊆ set ys ⟹ max-list xs ≤ max-list ys›
  using max-list-max max-list-in-Cons
  by (metis less-nat-zero-code linorder-not-le max-list.simps(1) subset-code(1))

lemma consistent-witness:
  fixes p :: ‹('f, 'p) fm›
  assumes ‹consistent S› and ‹p ∈ S› and ‹vars S ⊆ used› and ‹finite used›
  shows ‹consistent (witness used p ∪ S)›
  using assms
proof (induct used p rule: witness.induct)
  case (1 used p)
  moreover have ‹∃ n. max-set used < n›
    by blast
   ultimately obtain n where n: ‹witness used (¬ ∀ p) = {¬ p⟨#n/0⟩}› and
‹max-set used < n›
    by (metis someI-ex witness.simps(1))
  then have ‹max-set (vars S) < n›
    using 1(3−4) max-list-mono order-le-less-trans
   by (metis (no-types, lifting) Max.subset-imp bot.extremum-uniqueI less-nat-zero-code
linorder-neqE-nat max-set-def)
  moreover have ‹finite (vars S)›
    using 1(3−4) infinite-super by blast
  ultimately show ?case
    using 1 n(1) consistent-add-witness by metis
qed (auto simp: assms)

lemma consistent-extend:
  fixes f :: ‹nat ⇒ ('f, 'p) fm›
  assumes ‹consistent S› ‹finite (vars S)›
  shows ‹consistent (extend S f n)›
  using assms
proof (induct n)
  case (Suc n)
  then show ?case
    using consistent-witness[where S=‹{f n} ∪ -›] by (auto simp: Let-def)
qed simp

lemma consistent-Extend:
  fixes f :: ‹nat ⇒ ('f, 'p) fm›
  assumes ‹consistent S› ‹finite (vars S)›
  shows ‹consistent (Extend S f)›
  unfolding consistent-def
proof
  assume ‹∃ S'. set S' ⊆ Extend S f ∧ S' ⊢ ⊥›
  then obtain S' where ‹S' ⊢ ⊥› and ‹set S' ⊆ Extend S f›
    unfolding consistent-def by blast
  then obtain m where ‹set S' ⊆ (⋃ n ≤ m. extend S f n)›
    unfolding Extend-def using UN-finite-bound by (metis List.finite-set)
```

then have ‹*set* S' ⊆ *extend S f m*›
  using *extend-bound* **by** *blast*
moreover have ‹*consistent* (*extend S f m*)›
  using *assms consistent-extend* **by** *blast*
ultimately show *False*
  unfolding *consistent-def* **using** ‹$S' \vdash \bot$› **by** *blast*
qed

25 Maximal

definition ‹*maximal S* ≡ ∀ *p*. *p* ∉ *S* ⟶ ¬ *consistent* ({*p*} ∪ *S*)›

lemma *maximal-exactly-one*:
  assumes ‹*consistent S*› **and** ‹*maximal S*›
  shows ‹*p* ∈ *S* ⟷ (¬ *p*) ∉ *S*›
proof
  assume ‹*p* ∈ *S*›
  show ‹(¬ *p*) ∉ *S*›
  proof
    assume ‹(¬ *p*) ∈ *S*›
    then have ‹*set* [*p*, ¬ *p*] ⊆ *S*›
      using ‹*p* ∈ *S*› **by** *simp*
    moreover have ‹[*p*, ¬ *p*] ⊢ ⊥›
      by *blast*
    ultimately show *False*
      using ‹*consistent S*› **unfolding** *consistent-def* **by** *blast*
  qed
next
  assume ‹(¬ *p*) ∉ *S*›
  then have ‹¬ *consistent* ({¬ *p*} ∪ *S*)›
    using ‹*maximal S*› **unfolding** *maximal-def* **by** *blast*
  then obtain S' **where** ‹*set* S' ⊆ *S*› ‹(¬ *p*) # $S' \vdash \bot$›
    using ‹*consistent S*› *inconsistent-fm* **by** *blast*
  then have ‹$S' \vdash p$›
    using *Boole* **by** *blast*
  have ‹*consistent* ({*p*} ∪ *S*)›
    unfolding *consistent-def*
  proof
    assume ‹∃ S'. *set* S' ⊆ {*p*} ∪ *S* ∧ $S' \vdash \bot$›
    then obtain S'' **where** ‹*set* S'' ⊆ *S*› **and** ‹*p* # $S'' \vdash \bot$›
      using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *blast*
    then have ‹S' @ $S'' \vdash \bot$›
      using ‹$S' \vdash p$› **by** (*metis MP' add-imply imply.simps*(*2*) *imply-append*)
    moreover have ‹*set* (S' @ S'') ⊆ *S*›
      using ‹*set* S' ⊆ *S*› ‹*set* S'' ⊆ *S*› **by** *simp*
    ultimately show *False*
      using ‹*consistent S*› **unfolding** *consistent-def* **by** *blast*
  qed
  then show ‹*p* ∈ *S*›

using ‹*maximal S*› **unfolding** *maximal-def* **by** *blast*
qed

lemma *maximal-Extend*:
assumes ‹*surj f*›
shows ‹*maximal* (*Extend S f*)›
proof (*rule ccontr*)
assume ‹¬ *maximal* (*Extend S f*)›
then obtain *p* **where**
‹$p \notin$ *Extend S f*› **and** ‹*consistent* ({*p*} ∪ *Extend S f*)›
unfolding *maximal-def* **using** *assms consistent-Extend* **by** *blast*
obtain *k* **where** *k*: ‹*f k* = *p*›
using ‹*surj f*› **unfolding** *surj-def* **by** *metis*
then have ‹$p \notin$ *extend S f* (*Suc k*)›
using ‹$p \notin$ *Extend S f*› **unfolding** *Extend-def* **by** *blast*
then have ‹¬ *consistent* ({*p*} ∪ *extend S f k*)›
using *k* **by** (*auto simp*: *Let-def*)
moreover have ‹{*p*} ∪ *extend S f k* ⊆ {*p*} ∪ *Extend S f*›
unfolding *Extend-def* **by** *blast*
ultimately have ‹¬ *consistent* ({*p*} ∪ *Extend S f*)›
unfolding *consistent-def* **by** *auto*
then show *False*
using ‹*consistent* ({*p*} ∪ *Extend S f*)› **by** *blast*
qed

26 Saturation

definition ‹*saturated S* ≡ ∀ *p*. (¬ ∀ *p*) ∈ *S* ⟶ (∃ *n*. (¬ *p*⟨#*n*/*0*⟩) ∈ *S*)›

lemma *saturated-Extend*:
assumes ‹*consistent* (*Extend S f*)› **and** ‹*surj f*›
shows ‹*saturated* (*Extend S f*)›
proof (*rule ccontr*)
assume ‹¬ *saturated* (*Extend S f*)›
then obtain *p* **where** *p*: ‹(¬ ∀ *p*) ∈ *Extend S f*› ‹∄ *n*. (¬ *p*⟨#*n*/*0*⟩) ∈ *Extend S f*›
unfolding *saturated-def* **by** *blast*
obtain *k* **where** *k*: ‹*f k* = (¬ ∀ *p*)›
using ‹*surj f*› **unfolding** *surj-def* **by** *metis*

have ‹*extend S f k* ⊆ *Extend S f*›
unfolding *Extend-def* **by** *auto*
then have ‹*consistent* ({¬ ∀ *p*} ∪ *extend S f k*)›
using *assms*(*1*) *p*(*1*) **unfolding** *consistent-def* **by** *blast*
then have ‹∃ *n*. *extend S f* (*Suc k*) = {¬ *p*⟨#*n*/*0*⟩} ∪ {¬ ∀ *p*} ∪ *extend S f k*›
using *k* **by** (*auto simp*: *Let-def*)
moreover have ‹*extend S f* (*Suc k*) ⊆ *Extend S f*›
unfolding *Extend-def* **by** *blast*
ultimately show *False*

```
    using p(2) by auto
qed
```

27 Hintikka

```
locale Hintikka =
  fixes H :: ‹('f, 'p) fm set›
  assumes
    NoFalsity: ‹⊥ ∉ H› and
    ImpP: ‹(p ⟶ q) ∈ H ⟹ p ∉ H ∨ q ∈ H› and
    ImpN: ‹(p ⟶ q) ∉ H ⟹ p ∈ H ∧ q ∉ H› and
    UniP: ‹∀ p ∈ H ⟹ ∀ t. p⟨t/0⟩ ∈ H› and
    UniN: ‹∀ p ∉ H ⟹ ∃ n. p⟨#n/0⟩ ∉ H›
```

27.1 Model Existence

```
abbreviation hmodel (‹⟦-⟧›) where ‹⟦H⟧ ≡ ⟦#, †, λP ts. Pre P ts ∈ H⟧›

lemma semantics-tm-id [simp]:
  ‹⦇#, †⦈ t = t›
  by (induct t) (auto cong: map-cong)

lemma semantics-tm-id-map [simp]: ‹map ⦇#, †⦈ ts = ts›
  by (auto cong: map-cong)

theorem Hintikka-model:
  assumes ‹Hintikka H›
  shows ‹p ∈ H ⟷ ⟦H⟧ p›
proof (induct p rule: wf-induct[where r=‹measure size-fm›])
  case 1
  then show ?case ..
next
  case (2 x)
  show ‹x ∈ H ⟷ ⟦H⟧ x›
  proof (cases x; safe)
    case Falsity
    assume ‹⊥ ∈ H›
    then have False
      using assms Hintikka.NoFalsity by fast
    then show ‹⟦H⟧ ⊥› ..
  next
    case Falsity
    assume ‹⟦H⟧ ⊥›
    then have False
      by simp
    then show ‹⊥ ∈ H› ..
  next
    case (Pre P ts)
    assume ‹‡P ts ∈ H›
```

then show ‹$[\![H]\!]\ (\ddagger P\ ts)$›
  by *simp*
next
  case $(Pre\ P\ ts)$
  assume ‹$[\![H]\!]\ (\ddagger P\ ts)$›
  then show ‹$\ddagger P\ ts \in H$›
    by *simp*
next
  case $(Imp\ p\ q)$
  assume ‹$(p \longrightarrow q) \in H$›
  then have ‹$p \notin H \lor q \in H$›
    using *assms Hintikka.ImpP* **by** *blast*
  then have ‹$\neg\ [\![H]\!]\ p \lor [\![H]\!]\ q$›
    using *2 Imp* **by** *simp*
  then show ‹$[\![H]\!]\ (p \longrightarrow q)$›
    by *simp*
next
  case $(Imp\ p\ q)$
  assume ‹$[\![H]\!]\ (p \longrightarrow q)$›
  then have ‹$\neg\ [\![H]\!]\ p \lor [\![H]\!]\ q$›
    by *simp*
  then have ‹$p \notin H \lor q \in H$›
    using *2 Imp* **by** *simp*
  then show ‹$(p \longrightarrow q) \in H$›
    using *assms Hintikka.ImpN* **by** *blast*
next
  case $(Uni\ p)$
  assume ‹$\forall\, p \in H$›
  then have ‹$\forall\, t.\ p\langle t/0\rangle \in H$›
    using *assms Hintikka.UniP* **by** *metis*
  then have ‹$\forall\, t.\ [\![H]\!]\ (p\langle t/0\rangle)$›
    using *2 Uni* **by** *simp*
  then show ‹$[\![H]\!]\ (\forall\, p)$›
    by *simp*
next
  case $(Uni\ p)$
  assume ‹$[\![H]\!]\ (\forall\, p)$›
  then have ‹$\forall\, t.\ [\![H]\!]\ (p\langle t/0\rangle)$›
    by *simp*
  then have ‹$\forall\, t.\ p\langle t/0\rangle \in H$›
    using *2 Uni* **by** *simp*
  then show ‹$\forall\, p \in H$›
    using *assms Hintikka.UniN* **by** *blast*
qed
qed

27.2 Maximal Consistent Sets are Hintikka Sets

lemma *inconsistent-head*:

```
  assumes ‹consistent S› and ‹maximal S› and ‹p ∉ S›
  obtains S' where ‹set S' ⊆ S› and ‹p # S' ⊢ ⊥›
  using assms inconsistent-fm unfolding consistent-def maximal-def by metis

lemma inconsistent-parts [simp]:
  assumes ‹ps ⊢ ⊥› and ‹set ps ⊆ S›
  shows ‹¬ consistent S›
  using assms unfolding consistent-def by blast

lemma Hintikka-Extend:
  fixes H :: ‹('f, 'p) fm set›
  assumes ‹consistent H› and ‹maximal H› and ‹saturated H›
  shows ‹Hintikka H›
proof
  show ‹⊥ ∉ H›
  proof
    assume ‹⊥ ∈ H›
    moreover have ‹[⊥] ⊢ ⊥›
      by blast
    ultimately have ‹¬ consistent H›
      using inconsistent-parts[where ps=‹[⊥]›] by simp
    then show False
      using ‹consistent H› ..
  qed
next
  fix p q
  assume *: ‹(p ⟶ q) ∈ H›
  show ‹p ∉ H ∨ q ∈ H›
  proof safe
    assume ‹q ∉ H›
    then obtain Hq' where Hq': ‹q # Hq' ⊢ ⊥› ‹set Hq' ⊆ H›
      using assms inconsistent-head by metis

    assume ‹p ∈ H›
    then have ‹(¬ p) ∉ H›
      using assms maximal-exactly-one by blast
    then obtain Hp' where Hp': ‹(¬ p) # Hp' ⊢ ⊥› ‹set Hp' ⊆ H›
      using assms inconsistent-head by metis

    let ?H' = ‹Hp' @ Hq'›
    have H': ‹set ?H' = set Hp' ∪ set Hq'›
      by simp
    then have ‹set Hp' ⊆ set ?H'› and ‹set Hq' ⊆ set ?H'›
      by blast+
    then have ‹(¬ p) # ?H' ⊢ ⊥› and ‹q # ?H' ⊢ ⊥›
      using Hp'(1) Hq'(1) deduct imply-weaken by metis+
    then have ‹(p ⟶ q) # ?H' ⊢ ⊥›
      using Boole imply-Cons imply-head MP' cut by metis
    moreover have ‹set ((p ⟶ q) # ?H') ⊆ H›
```

using ‹$q \notin H$› $*(1)$ H' $Hp'(2)$ $Hq'(2)$ **by** *auto*
ultimately show *False*
using *assms* **unfolding** *consistent-def* **by** *blast*
qed
next
fix p q
assume $*$: ‹$(p \longrightarrow q) \notin H$›
show ‹$p \in H \wedge q \notin H$›
proof (*safe*, *rule ccontr*)
assume ‹$p \notin H$›
then obtain H' **where** S': ‹$p \# H' \vdash \bot$› ‹*set* $H' \subseteq H$›
using *assms inconsistent-head* **by** *metis*
moreover have ‹$(\neg (p \longrightarrow q)) \# H' \vdash p$›
by *auto*
ultimately have ‹$(\neg (p \longrightarrow q)) \# H' \vdash \bot$›
by *blast*
moreover have ‹*set* $((\neg (p \longrightarrow q)) \# H') \subseteq H$›
using $*(1)$ $S'(2)$ *assms maximal-exactly-one* **by** *auto*
ultimately show *False*
using *assms* **unfolding** *consistent-def* **by** *blast*
next
assume ‹$q \in H$›
then have ‹$(\neg q) \notin H$›
using *assms maximal-exactly-one* **by** *blast*
then obtain H' **where** H': ‹$(\neg q) \# H' \vdash \bot$› ‹*set* $H' \subseteq H$›
using *assms inconsistent-head* **by** *metis*
moreover have ‹$(\neg (p \longrightarrow q)) \# H' \vdash \neg q$›
by *auto*
ultimately have ‹$(\neg (p \longrightarrow q)) \# H' \vdash \bot$›
by *blast*
moreover have ‹*set* $((\neg (p \longrightarrow q)) \# H') \subseteq H$›
using $*(1)$ $H'(2)$ *assms maximal-exactly-one* **by** *auto*
ultimately show *False*
using *assms* **unfolding** *consistent-def* **by** *blast*
qed
next
fix p
assume ‹$\forall p \in H$›
then show ‹$\forall t.\ p\langle t/0\rangle \in H$›
using *assms consistent-add-instance* **unfolding** *maximal-def* **by** *blast*
next
fix p
assume ‹$\forall p \notin H$›
then show ‹$\exists n.\ p\langle \# n/0\rangle \notin H$›
using *assms maximal-exactly-one* **unfolding** *saturated-def* **by** *fast*
qed

28 Countable Formulas

instance *tm* :: (*countable*) *countable*
  by *countable-datatype*

instance *fm* :: (*countable*, *countable*) *countable*
  by *countable-datatype*

29 Completeness

theorem *strong-completeness*:
  fixes p :: ‹($'f$ :: *countable*, $'p$ :: *countable*) *fm*›
  assumes ‹$\forall$ (E :: - $\Rightarrow$ $'f$ *tm*) F G. *Ball* X $[\![E, F, G]\!]$ $\longrightarrow$ $[\![E, F, G]\!]$ p›
    and ‹*finite* (*vars* X)›
  shows ‹$\exists$ *ps*. *set ps* $\subseteq X \wedge$ *ps* $\vdash p$›
proof (*rule ccontr*)
  assume ‹$\nexists$ *ps*. *set ps* $\subseteq X \wedge$ *ps* $\vdash p$›
  then have $*$: ‹$\nexists$ *ps*. *set ps* $\subseteq X \wedge (\neg\ p)$ # *ps* $\vdash \bot$›
    using *Boole* **by** *blast*

  let *?S* = ‹$\{\neg\ p\} \cup X$›
  let *?H* = ‹*Extend ?S from-nat*›

  have ‹*consistent ?S*›
    using $*$ **by** (*metis consistent-def imply-Cons inconsistent-fm*)
  moreover have ‹*finite* (*vars ?S*)›
    using *assms* **by** *simp*
  ultimately have ‹*consistent ?H*› **and** ‹*maximal ?H*›
    using *assms consistent-Extend maximal-Extend surj-from-nat* **by** *blast+*
  moreover from *this* **have** ‹*saturated ?H*›
    using *saturated-Extend* **by** *fastforce*
  ultimately have ‹*Hintikka ?H*›
    using *assms Hintikka-Extend* **by** *blast*

  have ‹$[\![$*?H*$]\!]$ p› **if** ‹$p \in$ *?S*› **for** p
    using *that Extend-subset Hintikka-model* ‹*Hintikka ?H*› **by** *blast*
  then have ‹$[\![$*?H*$]\!]$ $(\neg\ p)$› **and** ‹$\forall q \in X$. $[\![$*?H*$]\!]$ q›
    by *fastforce+*
  moreover from *this* **have** ‹$[\![$*?H*$]\!]$ p›
    using *assms*(*1*) **by** *blast*
  ultimately show *False*
    by *simp*
qed

theorem *completeness*:
  fixes p :: ‹($'f$ :: *countable*, $'p$ :: *countable*) *fm*›
  assumes ‹$\forall$ (E :: - $\Rightarrow$ $'f$ *tm*) F G. $[\![E, F, G]\!]$ p›
  shows ‹$\vdash p$›
  using *assms strong-completeness*[**where** X=‹{}›] **by** *simp*

corollary
fixes p :: ‹(*unit*, *unit*) *fm*›
assumes ‹$\forall$ (E :: *nat* $\Rightarrow$ *unit tm*) F G. $[\![E, F, G]\!]$ p›
shows ‹$\vdash$ p›
using *completeness assms* **.**

30 Main Result

abbreviation *valid* :: ‹(*nat*, *nat*) *fm* $\Rightarrow$ *bool*› **where**
‹*valid* $p \equiv \forall$ (E :: *nat* $\Rightarrow$ *nat tm*) F G. $[\![E, F, G]\!]$ p›

theorem *main*: ‹*valid* $p \longleftrightarrow$ ($\vdash$ p)›
using *completeness soundness* **by** *blast*

end

References

[1] L. Henkin. The discovery of my completeness proofs. *Bulletin of Symbolic Logic*, 2(2):127–158, 1996.

[2] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.